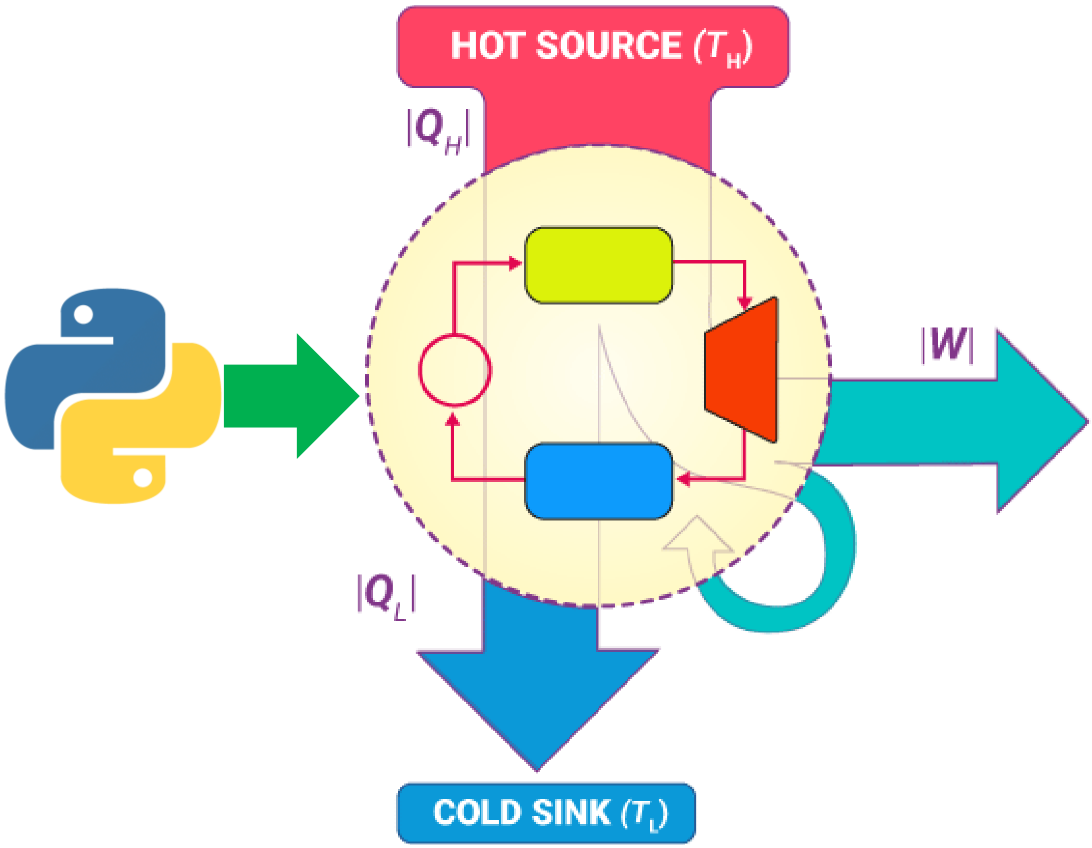
مبانی ترمودینامیک در پایتون

(معادلات حالت و مدل‌های ضریب فعالیت)

جلد اول



مترجم وتهیه کننده : علی زین الدینی

پیش گفتار

کتاب مبانی ترمودینامیک و پایتون (جلد اول) یکی از کتاب های مورد استفاده برای محققان شیمی و علاقه مندان حوزه برنامه نویسی پایتون است. در این کتاب سعی شده است تا مبانی ترمودینامیک در قالب کدهای برنامه نویسی پایتون در اختیار علاقه مندان قرار بگیرد. در جلد اول درمورد معادلات حالت و ضریب فعالیت بحث شده است و با استفاده از کتابخانه Thermo در پایتون این مباحث به زبان برنامه نویسی جهت محاسبات بهتر کد نویسی گردیده است.

تمامی مطالب استفاده شده ترجمه وب سایت زیر است، همچنین نحوه استفاده از این کتابخانه نیز شرح داده شده است.

https://thermo.readthedocs.io

این کتاب توسط علی زین الدینی، دانشجو رشته شیمی فیزیک و محقق در حوزه سیستم های شیمیایی و الکتروشیمیایی و توسعه دهنده پایتون تهیه گردیده است. برای ارتباط با مترجم و تهیه کننده میتوانید با ایمیل زیر مکاتبه داشته باشید. همچنین لینک صفحه اینستاگرامی ما نیز به آدرس زیر خواهد بود

Email: AliZeinodiny1996@gmail.com

Instagram: https://www.instagram.com/alizeinodiny1996_r

فهرست مطالب

نحوه نصب کتابخانه Thermo در پایتون

آخرین نسخه Thermo را از لینک زیر دریافت کنید

https://pypi.python.org/pypi/thermo/

اگر پایتون را با پیپ نصب کرده اید، به سادگی آن را با استفاده از:

```
$ pip install thermos
```

از طرف دیگر، اگر از conda به عنوان مدیریت بسته خود استفاده می کنید، می توانید به سادگی Thermo را از کانال conda-forge در محیط خود نصب کنید:

```
$ conda install -c conda-forge thermo
```

برای دریافت نسخه git، فرمان زیرا اجرا کنید

```
$ git clone git://github.com/CalebBell/thermo.git
```

آخرین کد منبع

آخرین نسخه توسعه منابع Thermo را می توانید در این آدرس دریافت کنید

https://github.com/CalebBell/thermo/issues

اطلاعات مجوز

برای اطلاعات در مورد شرایط و ضوابط استفاده از این نرم افزار، و سلب مسئولیت از کلیه ضمانت ها، به LICENSE.txt که آدرس آن در زیر قرار دارد مراجعه کنید. اگرچه Thermo نیازی به مجوز آن ندارد، اما اگر برای شما راحت است، لطفاً در صورت استفاده از Thermo در کار خود ذکر کنید. همچنین در نظر داشته باشید که هر تغییری که ایجاد می کنید برای جامعه و توسعه دهندگان مفید است.

https://github.com/CalebBell/thermo/blob/master/LICENSE.txt

گزارش اشکال

برای گزارش اشکالات، لطفاً از Thermo's Bug Tracker در آدرس زیر استفاده کنید:

https://github.com/CalebBell/thermo/issues

اگر سؤال بیشتری در مورد استفاده از کتابخانه دارید، با نویسنده در Caleb.Andrew.Bell@gmail.com تماس بگیرید.

نقل قول

برای استناد به Thermo در نشریات از عبارت زیر استفاده کنید:

```
Caleb Bell and Contributors (2016-2021). Thermo: Chemical properties
component of Chemical Engineering Design Library (ChEDL)
https://github.com/CalebBell/thermo.
```

شاخص ها و جداول

برای دسترسی به شاخص ها و جدول های این کتابخانه می‌توانید به آدرس های مشخص شده مراجعه کنید

شاخص ها

شاخص های ماژول

۱) مقدمه ای بر معادلات حالت مکعبی

معادلات حالت مکعبی، مدل‌های ترمودینامیکی سازگار و نسبتاً سریعی را برای مواد شیمیایی و مخلوط‌های خالص ارائه می‌کنند. آنها معمولاً برای نمایش گازها و مایعات استفاده می شوند.

فرم سه پارامتری عمومی به شرح زیر است:

$$P = \frac{RT}{V-b} - \frac{a\alpha(T)}{V^2 + \delta V + \epsilon}$$

این اساس پیاده سازی در ترمو را تشکیل می دهد. دو رابط مجزا ارائه شده است، thermo.eos برای مدل‌سازی جزء خالص و thermo.eos_mix برای مدل‌سازی چند جزئی. اجزای خالص کمی سریعتر از مخلوط های چند جزئی هستند، زیرا قوانین اختلاط واندروالس که به طور معمول مورد استفاده قرار می گیرند، عملیات N^2 را برای محاسبه α(T) انجام می دهند:

$$a\alpha = \sum_i \sum_j z_i z_j (a\alpha)_{ij}$$

سایر بخش‌های کندی که برای هر دو نوع اعمال می‌شود، محاسبه برخی ویژگی‌های اساسی (لیست در set_properties_from_solution است) است که سایر ویژگی‌ها ممکن است به آن‌ها بستگی داشته باشند، و محاسبه حجم مولی با توجه به یک جفت ورودی (T, P) (کل یک زیرماژول thermo.eos_volume) این موضوع را مورد بحث و اجرا قرار می دهد. هر دوی این محاسبات در زمان ثابت هستند، بنابراین سربار آنها برای اجزای خالص و مخلوط‌های چند جزیی یکسان است.

۱-۱) کار با اجزای خالص

ما می‌توانیم از رابط GCEOS (مخفف «معادله عمومی حالت مکعب ») با هر جزء یا معادله پیاده‌سازی‌شده حالت استفاده کنیم، اما برای سادگی، n -هگزان با Peng-Robinson EOS استفاده می‌شود. دمای بحرانی آن ۵۰۷,۶ کلوین، فشار بحرانی ۳,۰۲۵ مگاپاسکال و ضریب مرکز ۰,۲۹۷۵ است.

هنگام تنظیم اولیه یک شی GCEOS، وضعیت باید همراه با ثابت های بحرانی مشخص شود. ما در اینجا از ۴۰۰ K و 1e6 Pa استفاده می کنیم:

```
>>>from thermo import *
>>> eos = PR(Tc=507.6, Pc=3025000.0, omega=0.2975, T=400., P=1E6)
>>> eos PR(Tc=507.6, Pc=3025000.0, omega=0.2975, T=400.0, P=1000000.0)
```

رشته __repr__ طوری طراحی شده است که تمام ورودی های شی را نشان دهد. ما می توانیم راه حل های حجم را با ویژگی raw_volumes بررسی کنیم:

```
>>> eos.raw_volumes
(0.0001560731847856, 0.002141876816741, 0.000919295474982)
```

در این نقطه سه حجم واقعی وجود دارد، بنابراین یک محلول مایع مانند و یک محلول بخار مانند در دسترس است. صفت فاز در این حالت مقدار "l/g" خواهد داشت. در غیر این صورت "l" یا "g" خواهد بود.

```
>>> eos.phase
'l/g'
```

ویژگی های اساسی محاسبه شده در مقداردهی اولیه مستقیماً خواص هستند و می توان به آنها دسترسی داشت. خواص مایع مانند در انتهای نام خود "_l" دارند و "_g" در انتهای خواص گاز مانند است.

```
>>>eos.H_dep_l
-26111.877
>>> eos.S_dep_g
-6.4394518
>>> eos.dP_dT_l
288501.633
```

تمام محاسبات در thermo.eos و thermo.eos_mix بر اساس مولار است. وزن مولکولی هرگز ارائه یا مورد نیاز نیست. همه خروجی ها در واحدهای SI پایه (K، Pa، m^3، مول، و غیره) هستند. این توسعه را به طور قابل ملاحظه ای ساده کرد. برای کار با واحدهای مبتنی بر انبوه، از رابط فاز استفاده کنید. رابط‌های thermo.eos و thermo.eos_mix قبل از رابط فاز توسعه داده شده‌اند و دارای برخی ویژگی‌ها هستند که در اینترفیس فاز نمایش داده نشده‌اند. سایر ویژگی ها یا به عنوان روش هایی که به آرگومان نیاز دارند پیاده سازی می شوند، یا ویژگی های پایتون که دقیقاً مانند خواص‌ها عمل می کنند اما نتایج را به سرعت محاسبه می کنند. به عنوان مثال، فوگاسیتی فاز مایع fugacity_l یا ضریب انبساط هم‌باریک گاز (فشار ثابت) خواص هستند.

```
>>>eos.fugacity_l
421597.00785
>>> eos.beta_g
0.0101232239
```

تعداد بسیار زیادی از این ویژگی ها وجود دارد، زیرا بسیاری از آنها مشتقاتی هستند که شرایط مشابهی دارند. فهرست کامل در اسناد GCEOS موجود است. تماس‌های کمتری وجود دارد که دما را می‌گیرد، مانند Hvap که گرمای تبخیر جسم را در دمای مشخص محاسبه می‌کند:

```
>>> eos.Hvap(300)
31086.2
```

هنگامی که یک شی ایجاد شد، می توان از آن برای نمونه سازی اشیاء جدید GCEOS در شرایط مختلف بدون تعیین مجدد ثابت های بحرانی و سایر پارامترهای مورد نیاز استفاده کرد.

```
>>> eos.to(T=300.0, P=1e5)
PR(Tc=507.6, Pc=3025000.0, omega=0.2975, T=300.0, P=100000.0)
>>> eos.to(V=1e2, P=1e5)
PR(Tc=507.6, Pc=3025000.0, omega=0.2975, P=100000.0, V=100.0)
>>> eos.to(V=1e2, T=300)
PR(Tc=507.6, Pc=3025000.0, omega=0.2975, T=300, V=100.0)
```

همانطور که در مثال های بالا مشاهده شد، از هر دو T، P، V می توان برای مشخص کردن وضعیت جسم استفاده کرد. متغیرهای ورودی شی ذخیره می شوند و می توان آنها را با state_specs بررسی کرد:

```
>>> eos.state_specs
{'T': 400.0, 'P': 1000000.0}
```

بخش های جداگانه معادله عمومی مکعبی نیز ذخیره می شود. می توانیم از آنها برای بررسی اینکه معادله فشار برآورده شده است استفاده کنیم:

```
>>> from thermo.eos import R
>>> R*eos.T/(eos.V_l-eos.b) - eos.a_alpha/(eos.V_l**2 + eos.V_l*eos.delta +
 eos.epsilon)
1000000.000000
>>> R*eos.T/(eos.V_g-eos.b) - eos.a_alpha/(eos.V_g**2 + eos.V_g*eos.delta +
eos.epsilon)
1000000.000000
```

توجه داشته باشید که از آنجایی که نقاط شناور کاملاً دقیق نیستند، ممکن است برخی خطاهای کوچک نشان داده شود، اما دقت زیادی برای به حداقل رساندن آن انجام شده است.

مقدار ثابت گاز مورد استفاده J/(mol*K) ۸,۳۱۴۴۶۲۶۱۸۱۵۳۲۴ است. این نزدیک به دقت کامل اعداد ممیز شناور است، اما نه کاملاً. اکنون این یک مقدار دقیق است که به عنوان "تعریف" در سیستم SI استفاده می شود. توجه داشته باشید که سایر پیاده‌سازی‌های معادلات حالت ممکن است از مقدار کامل ثابت گاز استفاده نکنند، اما نویسنده اکیداً توصیه می‌کند هر کسی که پیاده‌سازی EOS خود را بنویسد از ثابت گاز کامل استفاده کند. این به نتایج قابل تعویض بیشتری اجازه می دهد.

۱-۲) تعادل اجزای خالص

در ادامه با همان حالت و مثال قبلی، دو راه حل از معادله حالت موجود بود. با این حال، مگر اینکه دمای دقیق ۴۰۰ کلوین و فشار 1 مگاپاسکال روی خط اشباع باشد، همیشه یک حالت ترمودینامیکی پایدار دیگر وجود دارد. ما باید از انرژی آزاد گیبس خروجی استفاده کنیم تا مشخص کنیم کدام حالت پایدارتر است. برای یک جزء خالص، حالتی که خروج انرژی آزاد گیبس را به حداقل می رساند، پایدارترین حالت است.

```
>>> eos = PR(Tc=507.6, Pc=3025000.0, omega=0.2975, T=400., P=1E6)
>>> eos.G_dep_l, eos.G_dep_g
(-2872.498434, -973.5198207)
```

به راحتی می توان دید فاز مایع پایدارتر است. این میانبر استفاده از انرژی آزاد گیبس خروجی فقط برای اجزای خالص با تمام فازهایی که از حالت مرجع گاز ایده آل استفاده می کنند معتبر است. معیار کامل این است که کدام حالت انرژی آزاد واقعی گیبس را به حداقل برساند. متد more_stable_phase این بررسی را انجام می دهد و "l" یا "g" را برمی گرداند:

```
>>> eos.more_stable_phase
'l'
```

برای یک جزء خالص، یک خط تعادل بخار-مایع درست تا نقطه بحرانی وجود دارد که فشار بخار سیال را تعیین می کند. این را می توان با استفاده از روش Psat محاسبه کرد:

```
>>> eos.Psat(400.0)
466205.073739
```

نتیجه تا بیش از ۱۰ رقم دقیق است و با استفاده از برخی تکنیک‌های ریاضی که امکان محاسبه مستقیم فشار بخار را فراهم می‌کند، اجرا می‌شود. چند رقم دیگر را می‌توان با تنظیم پولیش روی True به دست آورد، که نتیجه را با حل‌کننده نیوتن با دقتی که یک عدد ممیز شناور می‌تواند ارائه کند، پولیش می‌دهد:

```
>>> 1-eos.Psat(400, polish=True)/eos.Psat(400)
1.6e-14
```

چند روش دیگر مورد علاقه V_l_sat و V_g_sat هستند که مایع اشباع و حجم مولی را محاسبه می کنند. Tsat که دمای اشباع را با یک فشار مشخص محاسبه می کند و phi_sat که ضریب فوگاسیته اشباع را با توجه به یک دما محاسبه می کند.

```
>>> eos.V_l_sat(298.15), eos.V_g_sat(500)
(0.0001303559, 0.0006827569)
>>> eos.Tsat(101325.0)
341.76265
>>> eos.phi_sat(425.0)
0.8349716
```

۱-۳) کار با مخلوط ها

استفاده از مخلوط thermo.eos_mix ابتدا با استفاده از مخلوط هم‌مولار نیتروژن-متان در ۱۱۵ کلوین و ۱ مگاپاسکال و معادله پنگ-رابینسون نشان داده شده است:

```
>>> eos = PRMIX(T=115.0, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5, 46.04E5],
omegas=[0.04, 0.011], zs=[0.5, 0.5], kijs=[[0.0, 0.0289], [0.0289, 0.0]])
>>> eos.V_l, eos.V_g
(3.658707770e-05, 0.00070676607)
>>> eos.fugacities_l, eos.fugacities_g
([838516.99, 78350.27], [438108.61, 359993.48])
```

تمام ویژگی های موجود در GCEOS برای اشیاء GCEOSMIX نیز موجود است. اشیاء جدید GCEOSMIX را می توان با متد to ایجاد کرد که کسرهای مولی جدید zs و همچنین متغیرهای حالت جدید را می پذیرد. اگر یک ترکیب جدید zs ارائه نشده باشد، ترکیب فعلی نیز برای شی جدید استفاده می شود.

```
>>> eos.to(T=300.0, P=1e5)
PRMIX(Tcs=[126.1, 190.6], Pcs=[3394000.0, 4604000.0], omegas=[0.04, 0.011],
kijs=[[0.0, 0.0289], [0.0289, 0.0]], zs=[0.5, 0.5], T=300.0, P=100000.0)
>>> eos.to(T=300.0, P=1e5, zs=[.1, .9])
```

```
PRMIX(Tcs=[126.1, 190.6], Pcs=[3394000.0, 4604000.0], omegas=[0.04, 0.011],
kijs=[[0.0, 0.0289], [0.0289, 0.0]], zs=[0.1, 0.9], T=300.0, P=100000.0)
>>> eos.to(V=1, P=1e5, zs=[.4, .6])
PRMIX(Tcs=[126.1, 190.6], Pcs=[3394000.0, 4604000.0], omegas=[0.04, 0.011],
kijs=[[0.0, 0.0289], [0.0289, 0.0]], zs=[0.4, 0.6], P=100000.0, V=1)
>>> eos.to(V=1.0, T=300.0, zs=[.4, .6])
PRMIX(Tcs=[126.1, 190.6], Pcs=[3394000.0, 4604000.0], omegas=[0.04, 0.011],
kijs=[[0.0, 0.0289], [0.0289, 0.0]], zs=[0.4, 0.6], T=300.0, V=1.0)
```

ایجاد اشیاء جدید GCEOSMIX با متد زیرمجموعه که فقط از برخی از اجزای اولیه مشخص شده استفاده می کند امکان پذیر است:

```
>>> kijs = [[0.0, 0.00076, 0.00171], [0.00076, 0.0, 0.00061], [0.00171,
0.00061, 0.0]]
>>> PR3 = PRMIX(Tcs=[469.7, 507.4, 540.3], zs=[0.8168, 0.1501, 0.0331],
omegas=[0.249, 0.305, 0.349], Pcs=[3.369E6, 3.012E6, 2.736E6], T=322.29,
P=101325.0, kijs=kijs)
>>> PR3.subset([1,2])
PRMIX(Tcs=[507.4, 540.3], Pcs=[3012000.0, 2736000.0], omegas=[0.305,
0.349], kijs=[[0.0, 0.00061], [0.00061, 0.0]], zs=[0.8193231441048,
0.1806768558951], T=322.29, P=101325.0)
>>> PR3.subset([1,2], T=500.0, P=1e5, zs=[.2, .8])
PRMIX(Tcs=[507.4, 540.3], Pcs=[3012000.0, 2736000.0], omegas=[0.305,
0.349], kijs=[[0.0, 0.00061], [0.00061, 0.0]], zs=[0.2, 0.8], T=500.0,
P=100000.0)
>>> PR3.subset([1,2], zs=[.2, .8])
PRMIX(Tcs=[507.4, 540.3], Pcs=[3012000.0, 2736000.0], omegas=[0.305,
0.349], kijs=[[0.0, 0.00061], [0.00061, 0.0]], zs=[0.2, 0.8], T=322.29,
P=101325.0)
```

همچنین می توان اشیاء GCEOS خالص ایجاد کرد:

```
>>> PR3.pures()
[PR(Tc=469.7, Pc=3369000.0, omega=0.249, T=322.29, P=101325.0),
PR(Tc=507.4, Pc=3012000.0, omega=0.305, T=322.29, P=101325.0), PR(Tc=540.3,
Pc=2736000.0, omega=0.349, T=322.29, P=101325.0)]
```

دما، فشار، تعداد مول و مشتقات کسر مولی ضرایب فوگاسیتی ورود به سیستم نیز با روش‌های dlnphis_dT، dlnphis_dP، dlnphis_dns، و dlnphis_dzs در دسترس هستند:

```
>>> PR3.dlnphis_dT('l')
[0.029486952019, 0.03514175794, 0.040281845273]
>>> PR3.dlnphis_dP('l')
[-9.8253779e-06, -9.8189093031e-06, -9.8122598e-06]
>>> PR3.dlnphis_dns(PR3.Z_l)
[[-0.0010590517, 0.004153228837, 0.007300114797], [0.0041532288, -
0.016918292791, -0.0257680231], [0.0073001147, -0.02576802316, -
0.0632916462]]
>>> PR3.dlnphis_dzs(PR3.Z_l)
[[0.0099380692, 0.0151503498, 0.0182972357], [-0.038517738, -0.059589260, -
0.068438990], [-0.070571069, -0.103639207, -0.141162830]]
```

۱-۴) ویژگی های دیگر

هش کردن

با استفاده از متد model_hash می‌توان این دو شی را با یکدیگر مقایسه کرد و دید که آیا kijs، پارامترهای مدل و اجزای یکسانی دارند یا خیر:

```
>>> PR_case = PRMIX(T=115, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5,
46.04E5], omegas=[0.04, 0.011], zs=[0.5, 0.5], kijs=[[0,0.41],[0.41,0]])

>>> SRK_case = SRKMIX(T=115, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5,
46.04E5], omegas=[0.04, 0.011], zs=[0.5, 0.5], kijs=[[0,0.41],[0.41,0]])
```

```
>>> PR_case.model_hash() == SRK_case.model_hash()
False
```

با استفاده از متد state_hash می توان مشاهده کرد که آیا حالت دقیق و مدل بین دو شی مختلف مطابقت دارند یا خیر:

```
>>> PR_case2 = PRMIX(T=116, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5,
46.04E5], omegas=[0.04, 0.011], zs=[0.5, 0.5], kijs=[[0,0.41],[0.41,0]])
>>> PR_case.model_hash() == PR_case2.model_hash()
True
>>> PR_case.state_hash() == PR_case2.state_hash()
False
```

و در نهایت می توان با استفاده از روش __hash__ مشاهده کرد که آیا دو شی دقیقاً یکسان هستند، از جمله نتایج محاسبات کش شده:

```
>>> PR_case3 = PRMIX(T=115, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5,
46.04E5], omegas=[0.04, 0.011], zs=[0.5, 0.5], kijs=[[0,0.41],[0.41,0]])
>>> PR_case.state_hash() == PR_case3.state_hash()
True
>>> hash(PR_case) == hash(PR_case3)
True
>>> _ = PR_case.da_alpha_dT_ijs
>>> hash(PR_case) == hash(PR_case3)
False
```

سریال سازی

همه مدل‌های مکعبی EOS یک متد as_json و یک from_json را برای سریال‌سازی وضعیت شی برای انتقال از طریق شبکه، ذخیره‌سازی روی دیسک و ارسال داده‌ها بین فرآیندها ارائه می‌کنند.

```
>>> import json
>>> eos = PRSV2MIX(Tcs=[507.6], Pcs=[3025000], omegas=[0.2975], zs=[1],
T=299., P=1E6, kappa1s=[0.05104], kappa2s=[0.8634], kappa3s=[0.460])
>>> json_stuff = json.dumps(eos.as_json())
>>> new_eos = GCEOSMIX.from_json(json.loads(json_stuff))
>>> assert new_eos == eos
```

علاوه بر کتابخانه استاندارد json می‌توان از کتابخانه‌های دیگر json با طراحی استفاده کرد. ذخیره و بازآفرینی اشیا با کتابخانه pickle.dumps پایتون نیز آزمایش شده است. این می تواند سریعتر از استفاده از JSON به مقدار باینری بودن داده باشد.

۱-۵) تعادل مخلوط

بر خلاف اجزای خالص، تعیین وضعیت تعادل برای مخلوط ها ساده نیست. الگوریتم های مختلفی مانند جایگزینی متوالی و کمینه سازی گیبس استفاده می شود. همه آن‌ها نیاز به حدس‌های اولیه دارند، که معمولاً از مدل‌های ترمودینامیکی ساده‌تر می‌آیند. در حالی که در عمل می توان ترکیب تعادل را برای یک مسئله N فاز تعیین کرد، در تئوری باید از یک الگوریتم بهینه سازی جهانی استفاده شود.

جزئیات بیشتر در مورد این موضوع را می توانید در ماژول thermo.flash بیابید

۱-۶) استفاده از واحدها با معادلات حالت مکعبی

برای استفاده از این اشیا نیز یک بسته بندی پینتی وجود دارد.

```
>>> from thermo.units import *
>>> kwargs = dict(T=400.0*u.degC, P=30*u.psi, Tcs=[126.1, 190.6]*u.K,
Pcs=[33.94E5, 46.04E5]*u.Pa, omegas=[0.04, 0.011]*u.dimensionless, zs=[0.5,
0.5]*u.dimensionless, kijs=[[0.0, 0.0289], [0.0289, 0.0]]*u.dimensionless)
>>> eos_units = PRMIX(**kwargs)
>>> eos_units.H_dep_g, eos_units.T
(<Quantity(-2.53858854, 'joule / mole')>, <Quantity(673.15, 'kelvin')>)
```

```
>>> base = IG(T=300.0*u.K, P=1e6*u.Pa)
>>> base.V_g
<Quantity(0.00249433879, 'meter ** 3 / mole')>
```

۲) مقدمه ای بر مدل های ضریب فعالیت

سیستم های تعادل بخار-مایع و مایع-مایع می توانند رفتارهای متفاوتی داشته باشند. قانون رائول فقط می تواند وابستگی به دما و فشار را توصیف کند، بنابراین اغلب از یک ضریب تصحیح که وابستگی به ترکیب را اضافه می کند به نام "ضریب فعالیت" استفاده می شود. این یک رویکرد جداگانه برای استفاده از معادله حالت است، اما از آنجا که همبستگی فشار بخار مستقیم با ضرایب فعالیت استفاده می‌شود، می‌توان یک نتیجه با دقت بالاتر برای تعادل فاز به دست آورد. در حالی که این مدل‌ها اغلب «مدل‌های ضریب فعالیت» نامیده می‌شوند، در واقع یک پیش‌بینی برای انرژی اضافی گیبس هستند. ضرایب فعالیتی که برای تعادل فاز استفاده می شود از مشتق عدد مول جزئی انرژی گیبس اضافی بر اساس عبارت زیر بدست می آید:

$$\gamma_i = \exp\left(\frac{\frac{\partial n_i G^E}{\partial n_i}}{RT}\right)$$

۵ مدل ضریب فعالیت اساسی در ترمو وجود دارد:

1) NRTL
2) Wilson
3) UNIQUAC
4) Regular Solution
5) UNIFAC

هر یک از این مدل ها شی گرا هستند و از کلاس پایه GibbsExcess که بسیاری از روش های رایج را ارائه می دهد، به ارث می برند. یک کلاس ساختگی دیگر که صفر اضافی انرژی گیبس و ضرایب فعالیت ۱ را پیش بینی می کند، به عنوان IdealSolution موجود است. مدل انرژی اضافی گیبس معمولاً نسبتاً ساده است. تعدادی مشتق برای محاسبه سایر خصوصیات مانند ضریب فعالیت مورد نیاز است تا این عبارات پیچیده تر از مدل واقعی به نظر برسند. در ادبیات، معمول است که یک مدل به طور مستقیم به شکل ضریب فعالیت بدون بحث در مورد مدل انرژی اضافی گیبس نشان داده شود. برای نشان دادن تفاوت، در اینجا بیان انرژی گیبس مدل NRTL و مدل ضریب فعالیت آن آمده است:

$$g^E = RT\sum_i x_i \frac{\sum_j \tau_{ji} G_{ji} x_j}{\sum_j G_{ji} x_j}$$

$$\ln(\gamma_i) = \frac{\sum_{j=1}^{n} x_j \tau_{ji} G_{ji}}{\sum_{k=1}^{n} x_k G_{ki}} + \sum_{j=1}^{n} \frac{x_j G_{ij}}{\sum_{k=1}^{n} x_k G_{kj}} \left(\tau_{ij} - \frac{\sum_{m=1}^{n} x_m \tau_{mj} G_{mj}}{\sum_{k=1}^{n} x_k G_{kj}}\right)$$

مدل های NRTL، Wilson و UNIQUAC بیشترین استفاده را دارند. هر یک از آنها مبتنی بر رگرسیون هستند. هر یک از این مدل ها علاوه بر وابستگی به ترکیب، پارامترهای وابستگی به دما گسترده ای دارند. وابستگی‌های دمایی اعمال‌شده باید اجازه دهد که پارامترهای بسیاری از منابع دیگر در اینجا با آنها استفاده شوند.

مدل RegularSolution مبتنی بر مفهوم پارامتر حلالیت است. با حجم مولی مایع و پارامترهای حلالیت، یک مدل پیش بینی است. وابستگی به دما را نشان نمی دهد. ضرایب رگرسیون اضافی نیز می تواند با آن مدل استفاده شود.

مدل UNIFAC یک طرح مشارکت گروهی پیش‌بینی‌کننده است. در آن، هر مولکول به بخش های مختلف تقسیم می شود. این بخش ها دارای پارامترهای تعاملی با بخش های دیگر هستند. معمولا تکه تکه شدن با دست انجام نمی شود. یکی از ابزارهای آنلاین برای انجام این کار، DDBST Online Group Assignment Tool است.

۲-۱) ساختار شی

شی GibbsExcess چیزی در مورد تعادل فاز، فشار بخار، یا روال فلاش نمی داند. دامنه آن به مقابله با انرژی اضافی گیبس محدود است. به دلیل این مدولار بودن، یک شی GibbsExcess مقداردهی اولیه طراحی شده است تا در یک آرگومان به معادلات مکعب حالتی که از انرژی گیبس اضافی مانند PSRK استفاده می کند، ارسال شود.

مکان دیگری که این اشیاء مورد استفاده قرار می گیرند در اشیاء GibbsExcessLiquid است که قطعات را به هم نزدیک می کند تا یک فاز ترمودینامیکی (عمدتا) سازگار ایجاد کند که الگوریتم های فلاش می توانند با آن کار کنند. این ماژولار بودن به مدل‌های اضافی گیبس اجازه می‌دهد تا در هر مکانی نوشته و استفاده شوند، بنابراین مدل PSRK با خوشحالی به یک شی UNIFAC اجازه می‌دهد که مانند VTPR پیکربندی شده باشد.

۲-۲) مثال UNIFAC

مدل UNIFAC یک مدل پیش‌بینی مبتنی بر مشارکت گروهی است که با استفاده از «تکه‌بندی» هر مولکول به تعدادی «گروه» و «شمارش» آن‌ها کار می‌کند.

DDBST نمونه مشکلات متعددی را با استفاده از UNIFAC منتشر کرده است. یک سیستم دوتایی ساده از مثال P05.22a در ۲ با n-هگزان و بوتانون-۲ در زیر نشان داده شده است:

```
>>> from thermo.unifac import UFIP, UFSG, UNIFAC
>>> GE = UNIFAC.from_subgroups(chemgroups=[{1:2, 2:4}, {1:1, 2:1, 18:1}],
T=60+273.15, xs=[0.5, 0.5], version=0, interaction_data=UFIP,
subgroups=UFSG)
```

راه حل ارائه شده توسط DDBST دارای مقادیر ضریب فعالیت [۱,۴۲۸، ۱,۳۶۵] است که با موارد محاسبه شده توسط شی UNIFAC مطابقت دارد:

```
>>> GE.gammas()
[1.4276025835, 1.3646545010]
```

بسیاری از ویژگی های دیگر نیز پیاده سازی شده اند که تعدادی از آنها در زیر نشان داده شده است:

```
>>> GE.GE(), GE.dGE_dT(), GE.d2GE_dT2()
(923.641197, 0.206721488, -0.00380070204)
>>> GE.HE(), GE.SE(), GE.dHE_dT(), GE.dSE_dT()
(854.77193363, -0.2067214889, 1.266203886, 0.0038007020460)
```

توجه داشته باشید که متغیرهای UFIP و UFSG حاوی پارامترهای تعامل واقعی هستند. هیچ کدام با کلاس هاردکد نشده اند، بنابراین کلاس می تواند برای رگرسیون استفاده شود. پارامتر نسخه کنترل می کند که کدام نوع از UNIFAC استفاده شود، زیرا تعداد کمی از آنها وجود دارد. مدل‌های مختلف UNIFAC پیاده‌سازی شده شامل UNIFAC اصلی، UNIFAC دورتموند، PSRK، VTPR، Lyngby/Larsen، و UNIFAC KT است. پارامترهای تعامل برای همه مدل ها نیز گنجانده شده است، اما آرگومان نسخه به فایل های داده متصل نیست.

برای راحتی، تعدادی از تکه تکه شدن مولکول ها با کد UNIFAC توزیع می شوند. تمام قطعه بندی ها از طریق پورتال آنلاین DDBST، که در آن فایل های ساختار مولکولی را می توان ارسال کرد، به دست می‌آید و این مزیت را دارد که آنچه ارائه می شود بدون ابهام است. هیچ نگرانی در مورد اعداد CAS وجود ندارد، مانند اینکه چگونه گرافیت و الماس شماره CAS متفاوتی دارند در حالی که یک عنصر هستند یا هوا با وجود مخلوط بودن، دارای شماره CAS است. بر این اساس، فهرست موجود در این فایل‌های داده توزیع‌شده، کلیدهای InChI هستند که می‌توان آنها را از chemicals.identifiers یا در مکان‌های مختلف آنلاین دریافت کرد.

```
>>> import thermo.unifac
>>> thermo.unifac.load_group_assignments_DDBST()
>>> len(thermo.unifac.DDBST_UNIFAC_assignments)
28846
>>> len(thermo.unifac.DDBST_MODIFIED_UNIFAC_assignments)
29271
>>> len(thermo.unifac.DDBST_PSRK_assignments)
30034
>>> from chemicals import search_chemical
>>> search_chemical('toluene').InChI_key
'YXFVVABEGXRONW-UHFFFAOYSA-N'
```

```
>>> thermo.unifac.DDBST_MODIFIED_UNIFAC_assignments['YXFVVABEGXRONW-UHFFFAOYSA-N']
{9: 5, 11: 1}
```

لطفاً توجه داشته باشید که عدد صحیح شناسایی در این عناصر {group: count} لزوماً در نسخه‌های مختلف UNIFAC یکسان نیست.

۲-۳) نکاتی در مورد عملکرد

مقداردهی اولیه شی برای اولین بار یک عملیات با کارایی بالا نیست زیرا باید بررسی های خاصی انجام شود و ساختارهای داده تنظیم شود. برخی از قطعات معادلات مدل اضافی گیبس ممکن است فقط به دما یا ترکیب بستگی داشته باشند، نه اینکه به هر دو بستگی داشته باشند. هر مدل روش to_T_xs را پیاده سازی می کند که باید برای ایجاد یک شی جدید در دما و/یا ترکیب جدید استفاده شود. طراحی شی برای محاسبه خواص و غیرقابل تغییر بودن است: محاسبات در دماها و ترکیبات جدید در یک شی جدید انجام می شود. همچنین توجه داشته باشید که رشته __repr__ برای هر مدل طوری طراحی شده است که امکان بازسازی بدون اتلاف مدل را فراهم می کند. این در هنگام ساختن موارد آزمایشی بسیار مفید است.

```
>>> GE.to_T_xs(T=400.0, xs=[.1, .9])
UNIFAC(T=400.0, xs=[0.1, 0.9], rs=[4.4998000000000005, 3.2479], qs=[3.856, 2.876], Qs=[0.848, 0.54, 1.488], vs=[[2, 1], [4, 1], [0, 1]], psi_abc=([[0.0, 0.0, 476.4], [0.0, 0.0, 476.4], [26.76, 26.76, 0.0]], [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]], [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), version=0)
```

هنگام کار با تعداد کمی از مؤلفه ها (۵ یا کمتر)، PyPy بهترین عملکرد را ارائه می دهد و استفاده از مدل با لیست های Python به عنوان ورودی سریع ترین راه برای انجام محاسبات حتی در CPython است. اگر با بسیاری از مؤلفه ها کار کنید یا اگر آرایه های Numpy به عنوان ورودی و خروجی مورد نظر هستند، آرایه های Numpy می توانند به عنوان ورودی ارائه شوند. این تأثیر منفی بر عملکرد خواهد داشت مگر اینکه از رابط numba استفاده شود:

```
>>> import numpy as np
>>> import thermo.numba
>>> N = 3
>>> T = 25.0 + 273.15
>>> xs = np.array([0.7273, 0.0909, 0.1818])
>>> rs = np.array([.92, 2.1055, 3.1878])
>>> qs = np.array([1.4, 1.972, 2.4])
```

```
>>> tausA = tausC = tausD = tausE = tausF = np.array([[0.0]*N for i in
range(N)])
>>> tausB = np.array([[0, -526.02, -309.64], [318.06, 0, 91.532], [-1325.1,
-302.57, 0]])
>>> ABCDEF = (tausA, tausB, tausC, tausD, tausE, tausF)
>>> from thermo import UNIQUAC
>>> GE2 = UNIQUAC(T=T, xs=xs, rs=rs, qs=qs, ABCDEF=ABCDEF)
>>> GE2.gammas()
array([ 1.57039333,  0.29482416, 18.11432905])
```

رابط numba سرعت کد را افزایش می دهد و امکان محاسبات با ده ها مؤلفه را فراهم می کند. رابط numba نیاز دارد که همه ورودی‌ها آرایه‌های numpy باشند و همه خروجی‌های آن نیز آرایه‌های numba هستند.

```
>>> GE3 = thermo.numba.UNIQUAC(T=T, xs=xs, rs=rs, qs=qs, ABCDEF=ABCDEF)
>>> GE3.gammas()
array([ 1.57039333,  0.29482416, 18.11432905])
```

به عنوان نمونه ای از مزایای عملکرد، محاسبه گامای UNIFAC ۲۰۰ جزء، ۱۰,۶ میلی ثانیه در CPython و ۳۱۸ میکروثانیه زمانی که توسط Numba شتاب می گیرد، طول می کشد. در این مورد PyPy ۶۶۴ میکرو ثانیه طول می کشد. هنگامی که همان معیار با ۱۰ مؤلفه انجام می شود، محاسبه در CPython ۳۸۷میکروثانیه، با numba ۸۸,۶میکرو ثانیه و با PyPy ۳۶,۲ میکرو ثانیه طول می کشد.

استفاده مجدد از روش to_T_xs می تواند بسیار مهم باشد. برای UNIFAC ، چندین اصطلاح فقط به دما بستگی دارد. اگر محاسبه ۲۰۰ مؤلفه با مواردی که قبلاً محاسبه شده تکرار شود، زمان‌بندی‌ها در CPython ۲۶.۳ میکرو ثانیه، با numba ۱۲۷میکرو ثانیه و با PyPy ۱۲۵میکروثانیه است.

۲-۴) ویژگی های دیگر

ضرایب فعالیت محدود کننده رقت نامحدود را می توان با فراخوانی به gammas_infinite_dilution به دست آورد.

```
>>> GE.gammas_infinite_dilution()
[3.5659995166, 4.32849696]
```

همه مدل‌های ضریب فعالیت یک متد as_json و یک from_json را برای سریال‌سازی وضعیت شی برای انتقال از طریق شبکه، ذخیره‌سازی روی دیسک و ارسال داده‌ها بین فرآیندها ارائه می‌کنند.

```
>>> from thermo import IdealSolution
>>> import json
>>> model = IdealSolution(T=300.0, xs=[.1, .2, .3, .4])
>>> json_view = model.as_json()
>>> json_str = json.dumps(json_view)
>>> model_copy = IdealSolution.from_json(json.loads(json_str))
>>> assert model_copy == model
```

علاوه بر کتابخانه استاندارد json می‌توان از کتابخانه‌های دیگر json با طراحی استفاده کرد. ذخیره و بازآفرینی اشیا با کتابخانه pickle.dumps پایتون نیز آزمایش شده است. این می تواند سریعتر از استفاده از JSON به مقدار باینری بودن داده باشد. همه مدل‌ها دارای یک روش __hash__ هستند که می‌توان از آن برای مقایسه مدل‌های مختلف استفاده کرد تا ببینیم آیا کاملاً یکسان هستند (از جمله اینکه چه مقادیری قبلاً محاسبه شده‌اند).

آنها همچنین دارای یک متد model_hash هستند که می‌توان از آن برای مقایسه مدل‌های مختلف استفاده کرد تا ببینیم آیا پارامترهای مدل یکسانی دارند یا خیر.

آنها همچنین دارای یک متد state_hash هستند که می‌توان از آن برای مقایسه مدل‌های مختلف استفاده کرد تا ببینیم آیا دما، ترکیب و پارامترهای مدل یکسانی دارند یا خیر.

۲-۵) هویت های ضریب فعالیت

مجموعه ای از معادلات مفید به شرح زیر است. برای اطلاعات بیشتر، به منابع مراجعه شود. هیچ منبعی حاوی همه این اطلاعات نیست.

$$h^E = -T\frac{\partial g^E}{\partial T} + g^E$$

$$\frac{\partial h^E}{\partial T} = -T\frac{\partial^2 g^E}{\partial T^2}$$

$$\frac{\partial h^E}{\partial x_i} = -T\frac{\partial^2 g^E}{\partial T \partial x_i} + \frac{\partial g^E}{\partial x_i}$$

$$s^E = \frac{h^E - g^E}{T}$$

$$\frac{\partial s^E}{\partial T} = \frac{1}{T}\left(\frac{-\partial g^E}{\partial T} + \frac{\partial h^E}{\partial T} - \frac{(G+H)}{T}\right)$$

$$\frac{\partial S^E}{\partial x_i} = \frac{1}{T}\left(\frac{\partial h^E}{\partial x_i} - \frac{\partial g^E}{\partial x_i}\right)$$

$$\frac{\partial \gamma_i}{\partial n_i} = \gamma_i\left(\frac{\frac{\partial^2 G^E}{\partial x_i \partial x_j}}{RT}\right)$$

$$\frac{\partial \gamma_i}{\partial T} = \left(\frac{\frac{\partial^2 nG^E}{\partial T \partial n_i}}{RT} - \frac{\frac{\partial n_i G^E}{\partial n_i}}{RT^2}\right)\exp\left(\frac{\frac{\partial n_i G^E}{\partial n_i}}{RT}\right)$$

منابع

[1] Poling, Bruce E., John M. Prausnitz, and John P. O'Connell. The Properties of Gases and Liquids. 5th edition. New York: McGraw-Hill Professional, 2000.

[2] Gmehling, Jürgen, Michael Kleiber, Bärbel Kolbe, and Jürgen Rarey. Chemical Thermodynamics for Process Simulation. John Wiley & Sons, 2019.

[3] Nevers, Noel de. Physical and Chemical Equilibrium for Chemical Engineers. 2nd edition. Wiley, 2012.

[4] Elliott, J., and Carl Lira. Introductory Chemical Engineering Thermodynamics. 2nd edition. Upper Saddle River, NJ: Prentice Hall, 2012.

[5] Walas, Dr Stanley M. Phase Equilibria in Chemical Engineering. Butterworth-Heinemann, 1985.